CHAPTER 5

Working with Text Files

IN THIS CHAPTER

Using **vim** and **vi** to edit text files

Searching for files

Searching in files

When the UNIX system, on which Linux was based, was created, most information was managed on the system in plain-text files. Thus, it was critical for users to know how to use tools for searching for and within plain-text files and to be able to change and configure those files.

Today, configuration of Linux systems can still be done by editing plain-text files. Whether you are modifying files in the `/etc` directory to configure a local service or editing Ansible inventory files to configure sets of host computers, plain-text files are still in common use for those tasks.

Before you can become a full-fledged system administrator, you need to be able to use a plain-text editor. The fact that most professional Linux servers don't even have a graphical interface available makes the need for editing of plain-text configuration files with a non-graphical text editor necessary.

After you know how to edit text files, you still might find it tough to figure out where the files are located that you need to edit. With commands such as `find`, you can search for files based on various attributes (filename, size, modification date, and ownership to name a few). With the `grep` command, you can search inside of text files to find specific search terms.

Editing Files with vim and vi

It's almost impossible to use Linux for any period of time and not need a text editor because, as noted earlier, most Linux configuration files are plain-text files that you will almost certainly need to change manually at some point.

If you are using a GNOME desktop, you can run `gedit` (type **`gedit`** into the Search box and press Enter, or select Applications ➪ Accessories ➪ gedit), which is fairly intuitive for editing text.

Linux® Bible, Tenth Edition. Christopher Negus.

You can also run a simple text editor called `nano` from the shell. However, most Linux shell users use either the `vi` or `emacs` command to edit text files.

The advantage of `vi` or `emacs` over a graphical editor is that you can use the command from any shell, character terminal, or character-based connection over a network (using `telnet` or `ssh`, for example)—no graphical interface is required. They each also contain tons of features, so you can continue to grow with them.

The following sections provide a brief tutorial on the `vi` text editor, which you can use to manually edit a text file from any shell. It also describes the improved versions of `vi` called `vim`. (If `vi` doesn't suit you, see the sidebar "Exploring Other Text Editors" for further options.)

The `vi` editor is difficult to learn at first, but after you know it, you never have to use a mouse or a function key—you can edit and move around quickly and efficiently within files just by using the keyboard.

Exploring Other Text Editors

Dozens of text editors are available for use with Linux. Some alternatives might be in your Linux distribution. You can try them out if you find `vi` to be too taxing. Here are some of the options:

`nano`: This popular, streamlined text editor is used with many bootable Linux systems and other limited-space Linux environments. For example, `nano` is available to edit text files during a Gentoo Linux install process.

`gedit`: The GNOME text editor runs on the desktop.

`jed`: This screen-oriented editor was made for programmers. Using colors, `jed` can highlight code that you create so that you can easily read the code and spot syntax errors. Use the Alt key to select menus to manipulate your text.

`joe`: The joe editor is similar to many PC text editors. Use control and arrow keys to move around. Press Ctrl+C to exit with no save or Ctrl+X to save and exit.

`kate`: This nice-looking editor comes in the `kdebase` package. It has lots of bells and whistles, such as highlighting for different types of programming languages and controls for managing word wrap.

`kedit`: This GUI-based text editor comes with the KDE desktop.

`mcedit`: In this editor, function keys help you get around, save, copy, move, and delete text. Like `jed` and `joe`, `mcedit` is screen oriented. It comes in the mc package in RHEL and Fedora.

`nedit`: This is an excellent programmer's editor. You need to install the optional `nedit` package to get this editor.

If you use `ssh` to log in to other Linux computers on your network, you can use any available text editor to edit files. If you use `ssh -X` to connect to the remote system, a GUI-based editor pops up on your local screen. When no GUI is available, you need a text editor that runs in the shell, such as `vi`, `jed`, or `joe`.

Starting with vi

Most often, you start `vi` to open a particular file. For example, to open a file called `/tmp/test`, enter the following command:

```
$ vi /tmp/test
```

If this is a new file, you should see something similar to the following:

```
□
~
~
~
~
~
"/tmp/test" [New File]
```

A blinking box at the top represents where your cursor is located. The bottom line keeps you informed about what is going on with your editing (here, you just opened a new file). In between, there are tildes (~) as filler because there is no text in the file yet. Now here's the intimidating part: There are no hints, menus, or icons to tell you what to do. To make it worse, you can't just start typing. If you do, the computer is likely to beep at you. (And some people complain that Linux isn't friendly.)

First, you need to know the two main operating modes: command and input. The `vi` editor always starts in command mode. Before you can add or change text in the file, you have to type a command (one or two letters, sometimes preceded by an optional number) to tell `vi` what you want to do. Case is important, so use uppercase and lowercase exactly as shown in the examples!

NOTE

On Red Hat Enterprise Linux, Fedora, and other Linux distributions, for regular users the `vi` command is aliased to run `vim`. If you type `alias vi`, you should see `alias vi='vim'`. The first obvious difference between `vi` and `vim` is that any known text file type, such as HTML, C code, or a common configuration file, appears in color. The colors indicate the structure of the file. Other features of `vim` that are not in `vi` include features such as visual highlighting and split-screen mode. By default, the root user doesn't have `vi` aliased to `vim`. If `vim` is not on your system, try installing the vim-enhanced package.

Adding text

To get into input mode, type an input command letter. To begin, type any of the following letters. When you are finished inputting text, press the Esc key (sometimes twice) to return to command mode. Remember the Esc key!

- **a**: The *add* command. With this command, you can input text that starts to the *right* of the cursor.
- **A**: The *add at end* command. With this command, you can input text starting at the end of the current line.

- `i`: The *insert* command. With this command, you can input text that starts to the *left* of the cursor.
- `I`: The *insert at beginning* command. With this command, you can input text that starts at the beginning of the current line.
- `o`: The *open below* command. This command opens a line below the current line and puts you in insert mode.
- `O`: The *open above* command. This command opens a line above the current line and puts you in insert mode.

Tip

When you are in insert mode, `-- INSERT --` appears at the bottom of the screen.

Type a few words, and press Enter. Repeat that a few times until you have a few lines of text. When you're finished typing, press Esc to return to command mode. Now that you have a file with some text in it, try moving around in your text with the keys or letters described in the next section.

Tip

Remember the Esc key! It always places you back into command mode. Remember that sometimes you must press Esc twice. For example, if you type a colon (`:`) to go into `ex` mode, you must press Esc twice to return to command mode.

Moving around in the text

To move around in the text, you can use the up, down, right, and left arrows. However, many of the keys for moving around are right under your fingertips when they are in typing position:

- **Arrow keys**: Move the cursor up, down, left, or right in the file one character at a time. To move left and right, you can also use Backspace and the spacebar, respectively. If you prefer to keep your fingers on the keyboard, move the cursor with h (left), l (right), j (down), or k (up).
- **w**: Moves the cursor to the beginning of the next word (delimited by spaces, tabs, or punctuation).
- **W**: Moves the cursor to the beginning of the next word (delimited by spaces or tabs).
- **b**: Moves the cursor to the beginning of the previous word (delimited by spaces, tabs, or punctuation).
- **B**: Moves the cursor to the beginning of the previous word (delimited by spaces or tabs).
- **0 (zero)**: Moves the cursor to the beginning of the current line.
- **$**: Moves the cursor to the end of the current line.
- **H**: Moves the cursor to the upper-left corner of the screen (first line on the screen).

M: Moves the cursor to the first character of the middle line on the screen.

L: Moves the cursor to the lower-left corner of the screen (last line on the screen).

Deleting, copying, and changing text

The only other editing that you need to know is how to delete, copy, or change text. The `x`, `d`, `y`, and `c` commands can be used to delete and change text. These can be used along with movement keys (arrows, PgUp, PgDn, letters, and special keys) and numbers to indicate exactly what you are deleting, copying, or changing. Consider the following examples:

x: Deletes the character under the cursor.

X: Deletes the character directly before the cursor.

d<?>: Deletes some text.

c<?>: Changes some text.

y<?>: Yanks (copies) some text.

The `<?>` after each letter in the preceding list identifies the place where you can use a movement command to choose what you are deleting, changing, or yanking. For example:

dw: Deletes (`d`) a word (`w`) after the current cursor position.

db: Deletes (`d`) a word (`b`) before the current cursor position.

dd: Deletes (`d`) the entire current line (`d`).

c$: Changes (`c`) the characters (actually erases them) from the current character to the end of the current line (`$`) and goes into input mode.

c0: Changes (`c`) (again, erases) characters from the previous character to the beginning of the current line (`0`) and goes into input mode.

cl: Erases (`c`) the current letter (`l`) and goes into input mode.

cc: Erases (`c`) the line (`c`) and goes into input mode.

yy: Copies (`y`) the current line (`y`) into the buffer.

y): Copies (`y`) the current sentence (`)`), to the right of the cursor, into the buffer.

y}: Copies (`y`) the current paragraph (`}`), to the right of the cursor, into the buffer.

Any of the commands just shown can be further modified using numbers, as you can see in the following examples:

3dd: Deletes (`d`) three (`3`) lines (`d`), beginning at the current line.

3dw: Deletes (`d`) the next three (`3`) words (`w`).

5cl: Changes (`c`) the next five (`5`) letters (`l`) (that is, removes the letters and enters input mode).

12j: Moves down (`j`) 12 lines (`12`).

5cw: Erases (`c`) the next five (`5`) words (`w`) and goes into input mode.

4y): Copies (`y`) the next four (`4`) sentences (`)`).

Pasting (putting) text

After text has been copied to the buffer (by deleting, changing, or yanking it), you can place that text back in your file using the letter `p` or `P`. With both commands, the text most recently stored in the buffer is put into the file in different ways.

`P`: Puts the copied text to the left of the cursor if the text consists of letters or words; puts the copied text above the current line if the copied text contains lines of text.

`p`: Puts the buffered text to the right of the cursor if the text consists of letters or words; puts the buffered text below the current line if the buffered text contains lines of text.

Repeating commands

After you delete, change, or paste text, you can repeat that action by typing a period (.). For example, with the cursor on the beginning of the name **`Joe`**, you type **`cw`** and then type **`Jim`** to change `Joe` to `Jim`. You search for the next occurrence of `Joe` in the file, position the cursor at the beginning of that name, and press a period. The word changes to `Jim`, and you can search for the next occurrence. You can go through a file this way, pressing `n` to go to the next occurrence and period (.) to change the word.

Exiting vi

To wrap things up, use the following commands to save or quit the file:

`ZZ`: Saves the current changes to the file and exits from `vi`.

`:w`: Saves the current file, but you can continue editing.

`:wq`: Works the same as `ZZ`.

`:q`: Quits the current file. This works only if you don't have any unsaved changes.

`:q!`: Quits the current file and *doesn't* save the changes you just made to the file.

Tip

If you've really trashed the file by mistake, the `:q!` command is the best way to exit and abandon your changes. The file reverts to the most recently changed version. So, if you just saved with `:w`, you are stuck with the changes up to that point. However, despite having saved the file, you can type `u` to back out of changes (all the way back to the beginning of the editing session if you like) and then save again.

You have learned a few `vi` editing commands. I describe more commands in the following sections. First, however, consider the following tips to smooth out your first trials with `vi`:

Esc: Remember that Esc gets you back to command mode. (I've watched people press every key on the keyboard trying to get out of a file.) Esc followed by `ZZ` gets you out of command mode, saves the file, and exits.

u: Press `u` to undo the previous change you made. Continue to press u to undo the change before that and the one before that.

Ctrl+R: If you decide that you didn't want to undo the previous undo command, use Ctrl+R for Redo. Essentially, this command undoes your undo.

Caps Lock: Beware of hitting Caps Lock by mistake. Everything that you type in `vi` has a different meaning when the letters are capitalized. You don't get a warning that you are typing capitals; things just start acting weird.

`:!command`: You can run a shell command while you are in `vi` using **`:!`** followed by a shell command name. For example, type **`:!date`** to see the current date and time, type **`:!pwd`** to see what your current directory is, or type **`:!jobs`** to see whether you have any jobs running in the background. When the command completes, press Enter and you are back to editing the file. You could even use this technique to launch a shell (`:!bash`) from `vi`, run a few commands from that shell, and then type `exit` to return to `vi`. (I recommend doing a save before escaping to the shell, just in case you forget to go back to `vi`.)

Ctrl+g: If you forget what you are editing, pressing these keys displays the name of the file that you are editing and the current line that you are on at the bottom of the screen. It also displays the total number of lines in the file, the percentage of how far you are through the file, and the column number the cursor is on. This just helps you get your bearings after you've stopped for a cup of coffee at 3 a.m.

Skipping around in the file

Besides the few movement commands described earlier, there are other ways of moving around a `vi` file. To try these out, open a large file that you can't damage too much. (Try copying `/var/log/messages` to `/tmp` and opening it in `vi`.) Here are some movement commands that you can use:

Ctrl+f: Pages ahead one page at a time.

Ctrl+b: Pages back one page at a time.

Ctrl+d: Pages ahead one-half page at a time.

Ctrl+u: Pages back one-half page at a time.

G: Goes to the last line of the file.

1G: Goes to the first line of the file.

35G: Goes to any line number (35, in this case).

Searching for text

To search for the next or previous occurrence of text in the file, use either the slash (/) or the question mark (?) character. Follow the slash or question mark with a pattern (string of text) to search forward or backward, respectively, for that pattern. Within the search, you can also use metacharacters. Here are some examples:

`/hello`: Searches forward for the word `hello`.

`?goodbye`: Searches backward for the word `goodbye`.

/The.*foot: Searches forward for a line that has the word `The` in it and also, after that at some point, the word `foot`.

?[pP]rint: Searches backward for either `print` or `Print`. Remember that case matters in Linux, so make use of brackets to search for words that could have different capitalization.

After you have entered a search term, simply type **n** or **N** to search again in the same direction (`n`) or the opposite direction (`N`) for the term.

Using ex mode

The `vi` editor was originally based on the `ex` editor, which didn't let you work in full-screen mode. However, it did enable you to run commands that let you find and change text on one or more lines at a time. When you type a colon and the cursor goes to the bottom of the screen, you are essentially in `ex` mode. The following are examples of some of those `ex` commands for searching for and changing text. (I chose the words `Local` and `Remote` to search for, but you can use any appropriate words.)

:g/Local: Searches for the word `Local` and prints every occurrence of that line from the file. (If there is more than a screenful, the output is piped to the `more` command.)

:s/Local/Remote: Substitutes `Remote` for the first occurrence of the word `Local` on the current line.

:g/Local/s//Remote: Substitutes the first occurrence of the word `Local` on every line of the file with the word `Remote`.

:g/Local/s//Remote/g: Substitutes every occurrence of the word `Local` with the word `Remote` in the entire file.

:g/Local/s//Remote/gp: Substitutes every occurrence of the word `Local` with the word `Remote` in the entire file and then prints each line so that you can see the changes (piping it through `less` if output fills more than one page).

Learning more about vi and vim

To learn more about the `vi` editor, try typing **vimtutor**. The `vimtutor` command opens a tutorial in the `vim` editor that steps you through common commands and features you can use in `vim`. To use `vimtutor`, install the `vim`-enhanced package.

Finding Files

Even a basic Linux installation can have thousands of files installed on it. To help you find files on your system, you can use commands such as `locate` (to find commands by name), `find` (to find files based on lots of different attributes), and `grep` (to search within text files to find lines in files that contain search text).

Using locate to find files by name

On most Linux systems (Fedora and RHEL included), the `updatedb` command runs once per day to gather the names of files throughout your Linux system into a database. By running the `locate` command, you can search that database to find the location of files stored in it.

Here are a few things that you should know about searching for files using the `locate` command:

- There are advantages and disadvantages to using `locate` to find filenames instead of the `find` command. A `locate` command finds files faster because it searches a database instead of having to search the whole filesystem live. A disadvantage is that the `locate` command cannot find any files added to the system since the previous time the database was updated.
- Not every file in your filesystem is stored in the database. The contents of the `/etc/updatedb.conf` file limit which filenames are collected by pruning out select mount types, filesystem types, file types, and mount points. For example, filenames are not gathered from remotely mounted filesystems (`cifs`, `nfs`, and so on) or locally mounted CDs or DVDs (`iso9660`). Paths containing temporary files (`/tmp`) and spool files (`/var/spool/cups`) are also pruned. You can add items to prune (or remove some items that you don't want pruned) the locate database to your needs. In RHEL 8, the `updatedb.conf` file contains the following:

```
PRUNE_BIND_MOUNTS = "yes"
PRUNEFS = "9p afs anon_inodefs auto autofs bdev binfmt_misc cgroup
cifs coda configfs cpuset debugfs devpts ecryptfs exofs fuse fuse
.sshfs fusectl gfs gfs2 gpfs hugetlbfs inotifyfs iso9660 jffs2
lustre mqueue ncpfs nfs nfs4 nfsd pipefs proc ramfs rootfs rpc_
pipefs securityfs selinuxfs sfs sockfs sysfs tmpfs ubifs udf usbfs
ceph fuse.ceph"
PRUNENAMES = ".git .hg .svn .bzr .arch-ids {arch} CVS"
PRUNEPATHS = "/afs /media /mnt /net /sfs /tmp /udev /var/cache/
ccache /var/lib/yum/yumdb /var/lib/dnf/yumdb /var/spool/cups /var/
spool/squid /var/tmp /var/lib/ceph"
```

As a regular user, you can't see any files from the locate database that you can't see in the filesystem normally. For example, if you can't type `ls` to view files in the `/root` directory, you can't locate files stored in that directory.

- When you search for a string, the string can appear anywhere in a file's path. For example, if you search for `passwd`, you could turn up `/etc/passwd`, `/usr/bin/passwd`, `/home/chris/passwd/pwdfiles.txt`, and many other files with `passwd` in the path.
- If you add files to your system after `updatedb` runs, you can't locate those files until `updatedb` runs again (probably that night). To get the database to contain all files up to the current moment, you can simply run `updatedb` from the shell as root.

Here are some examples of using the `locate` command to search for files:

```
$ locate .bashrc
/etc/skel/.bashrc
/home/cnegus/.bashrc
# locate .bashrc
/etc/skel/.bashrc
/home/bill/.bashrc
/home/joe/.bashrc
/root/.bashrc
```

When run as a regular user, `locate` only finds `.bashrc` in `/etc/skel` and the user's own home directory. Run as root, the same command locates `.bashrc` files in everyone's home directory.

```
$ locate dir_color
/usr/share/man/man5/dir_colors.5.gz
...
$ locate -i dir_color
/etc/DIR_COLORS
/etc/DIR_COLORS.256color
/etc/DIR_COLORS.lightbgcolor
/usr/share/man/man5/dir_colors.5.gz
```

Using locate `-i`, filenames are found regardless of case. So in the previous example, `DIR_COLORS` was found with `-i` whereas it wasn't found without the `-i` option.

```
$ locate services
/etc/services
/usr/share/services/bmp.kmgio
/usr/share/services/data.kmgio
```

Unlike the `find` command, which uses the `-name` option to find filenames, the `locate` command locates the string you enter if it exists in any part of the file's path. In this example, searching for `services` using the `locate` command finds files and directories containing the `services` text string.

Searching for files with find

The `find` command is the best one for searching your filesystem for files based on a variety of attributes. After files are found, you can act on those files as well (using the `-exec` or `-okay` option) by running any commands you want on them.

When you run `find`, it searches your filesystem live, which causes it to run slower than `locate`, but it gives you an up-to-the-moment view of the files on your Linux system. However, you can also tell `find` to start at a particular point in the filesystem so that the search can go faster by limiting the area of the filesystem being searched.

Nearly any file attribute that you can think of can be used as a search option. You can search for filenames, ownership, permission, size, modification times, and other attributes. You can even use combinations of attributes. Here are some basic examples of using the `find` command:

```
$ find
$ find /etc
# find /etc
$ find $HOME -ls
```

Run on a line by itself, the `find` command finds all files and directories below the current directory. If you want to search from a particular point in the directory tree, just add the name of the directory you want to search (such as `/etc`). As a regular user, `find` does not give you special permission to find files that have permissions that make them readable only by the root user. So, `find` produces a bunch of error messages. Run as the root user, `find /etc` finds all files under `/etc`.

A special option to the `find` command is `-ls`. A long listing (ownership, permission, size, and so on) is printed with each file when you add `-ls` to the `find` command (similar to output of the `ls -l` command). This option helps you in later examples when you want to verify that you have found files that contain the ownership, size, modification times, or other attributes that you are trying to find.

Note

If, as a regular user, you are searching an area of the filesystem where you don't have full permission to access all of the files it contains (such as the `/etc` directory), you might receive lots of error messages when you search with `find`. To get rid of those messages, direct standard errors to `/dev/null`. To do that, add the following to the end of the command line: `2> /dev/null`. The `2>` redirects standard error to the next option (in this case `/dev/null`, where the output is discarded).

Finding files by name

To find files by name, you can use the `-name` and `-iname` options. The search is done by base name of the file; the directory names are not searched by default. To make the search more flexible, you can use file-matching characters, such as asterisks (`*`) and question marks (`?`), as in the following examples:

```
# find /etc -name passwd
/etc/pam.d/passwd
/etc/passwd
# find /etc -iname '*passwd*'
/etc/pam.d/passwd
/etc/passwd-
/etc/passwd.OLD
/etc/passwd
/etc/MYPASSWD
/etc/security/opasswd
```

Using the `-name` option and no asterisks, the first example above lists any files in the `/etc` directory that are named `passwd` exactly. By using `-iname` instead, you can match any combination of upper- and lowercase. Using asterisks, you can match any filename that includes the word `passwd`.

Finding files by size

If your disk is filling up and you want to find out where your biggest files are located, you can search your system by file size. The `-size` option enables you to search for files that are exactly, smaller than, or larger than a selected size, as you can see in the following examples:

```
$ find /usr/share/ -size +10M
$ find /mostlybig -size -1M
$ find /bigdata -size +500M -size -5G -exec du -sh {} \;
4.1G    /bigdata/images/rhel6.img
606M    /bigdata/Fedora-16-i686-Live-Desktop.iso
560M    /bigdata/dance2.avi
```

The first example in the preceding code finds files larger than 10MB. The second finds files less than 1MB. In the third example, I'm searching for files that are between 500MB and 5GB. This includes an example of the `-exec` option (which I describe later) to run the `du` command on each file to see its size.

Finding files by user

You can search for a particular owner (`-user`) or group (`-group`) when you try to find files. By using `-not` and `-or`, you can refine your search for files associated with specific users and groups, as you can see in the following examples:

```
$ find /home -user chris -ls
131077    4 -rw-r--r--  1 chris   chris 379 Jun 29  2014 ./.bashrc
# find /home \( -user chris -or -user joe \) -ls
131077    4 -rw-r--r--  1 chris   chris 379 Jun 29  2014 ./.bashrc
181022    4 -rw-r--r--  1 joe     joe    379 Jun 15  2014 ./.bashrc
# find /etc -group ntp -ls
131438    4 drwxrwsr-x  3 root    ntp   4096 Mar  9 22:16 /etc/ntp
# find /var/spool -not -user root -ls
262100    0 -rw-rw----   1 rpc      mail    0 Jan 27  2014 /var/spool/mail/rpc
278504    0 -rw-rw----   1 joe      mail    0 Apr  3  2014 /var/spool/mail/joe
261230    0 -rw-rw----   1 bill     mail    0 Dec 18 14:17 /var/spool/mail/bill
277373 2848 -rw-rw----   1 chris    mail 8284 Mar 15  2014 /var/spool/mail/chris
```

The first example outputs a long listing of all of the files under the `/home` directory that are owned by the user `chris`. The next lists files owned by `chris` or `joe`. The `find` command of `/etc` turns up all files that have `ntp` as their primary group assignment. The last example shows all files under `/var/spool` that are not owned by root. You can see files owned by other users in the sample output.

Finding files by permission

Searching for files by permission is an excellent way to turn up security issues on your system or uncover access issues. Just as you changed permissions on files using numbers or letters (with the `chmod` command), you can likewise find files based on number or letter permissions along with the `-perm` options. (Refer to Chapter 4, "Moving Around the Filesystem," to see how to use numbers and letters with `chmod` to reflect file permissions.)

If you use numbers for permission, as I do below, remember that the three numbers represent permissions for the user, group, and other. Each of those three numbers varies from no permission (0) to full read/write/execute permission (7) by adding read (4), write (2), and execute (1) bits together. With a hyphen (-) in front of the number, all three of the bits indicated must match; with a forward slash (/) in front of it, any of the numbers can match for the search to find a file. The full, exact numbers must match if neither a hyphen nor a forward slash is used.

Consider the following examples:

```
$ find /usr/bin -perm 755 -ls
788884   28 -rwxr-xr-x   1 root     root        28176 Mar 10  2014 /bin/echo

$ find /home/chris/ -perm -222 -type d -ls
144503    4 drwxrwxrwx   8 chris  chris 4096 Jun 23  2014 /home/chris/OPENDIR
```

By searching for `-perm 755`, any files or directories with exactly `rwxr-xr-x` permission are matched. By using `-perm -222`, only files that have write permission for user, group, and other are matched. Notice that, in this case, the `-type d` is added to match only directories.

```
$ find /myreadonly -perm /222 -type f
685035    0 -rw-rw-r--  1 chris   chris      0 Dec 30 16:34 /myreadonly/abc

$ find . -perm -002 -type f -ls
266230    0 -rw-rw-rw-   1 chris   chris      0 Dec 30 16:28 ./LINUX_BIBLE/abc
```

Using `-perm /222`, you can find any file (`-type f`) that has write permission turned on for the user, group, or other. You might do that to make sure that all files are read-only in a particular part of the filesystem (in this case, beneath the `/myreadonly` directory). The last example, `-perm /002`, is very useful for finding files that have open write permission for "other," regardless of how the other permission bits are set.

Finding files by date and time

Date and time stamps are stored for each file when it is created, when it is accessed, when its content is modified, or when its metadata is changed. Metadata includes owner, group, time stamp, file size, permissions, and other information stored in the file's inode. You might want to search for file data or metadata changes for any of the following reasons:

- You just changed the contents of a configuration file, and you can't remember which one. So, you search `/etc` to see what has changed in the past 10 minutes:

```
$ find /etc/ -mmin -10
```

- You suspect that someone hacked your system three days ago. So, you search the system to see if any commands have had their ownership or permissions changed in the past three days:

```
$ find /bin /usr/bin /sbin /usr/sbin -ctime -3
```

- You want to find files in your FTP server (`/var/ftp`) and web server (`/var/www`) that have not been accessed in more than 300 days so that you can see if any need to be deleted:

```
$ find /var/ftp /var/www -atime +300
```

As you can glean from the examples, you can search for content or metadata changes over a certain number of days or minutes. The time options (`-atime`, `-ctime`, and `-mtime`) enable you to search based on the number of days since each file was accessed, changed, or had its metadata changed. The `min` options (`-amin`, `-cmin`, and `-mmin`) do the same in minutes.

Numbers that you give as arguments to the `min` and `time` options are preceded by a hyphen (to indicate a time from the current time to that number of minutes or days ago) or a plus (to indicate time from the number of minutes or days ago and older). With no hyphen or plus, the exact number is matched.

Using 'not' and 'or' when finding files

With the `-not` and `-or` options, you can further refine your searches. There may be times when you want to find files owned by a particular user but not assigned to a particular group. You may want files larger than a certain size but smaller than another size. Or you might want to find files owned by any of several users. The `-not` and `-or` options can help you do that. Consider the following examples:

- There is a shared directory called `/var/allusers`. This command line enables you to find files that are owned by either `joe` or `chris`.

```
$ find /var/allusers \( -user joe -o -user chris \) -ls
679967    0 -rw-r--r-- 1 chris chris    0 Dec 31 12:57
  /var/allusers/myjoe
679977 1812 -rw-r--r-- 1 joe   joe   4379 Dec 31 13:09
  /var/allusers/dict.dat
679972    0 -rw-r--r-- 1 joe   sales    0 Dec 31 13:02
  /var/allusers/one
```

- This command line searches for files owned by the user `joe`, but only those that are not assigned to the group `joe`:

```
$ find /var/allusers/ -user joe -not -group joe -ls
679972 0 -rw-r--r-- 1 joe sales  0 Dec 31 13:02 /var/allusers/one
```

- You can also add multiple requirements on your searches. Here, a file must be owned by the user `joe` and must also be more than 1MB in size:

```
$ find /var/allusers/ -user joe -and -size +1M -ls
679977 1812 -rw-r--r-- 1 joe root 1854379 Dec 31 13:09
  /var/allusers/dict.dat
```

Finding files and executing commands

One of the most powerful features of the `find` command is the capability to execute commands on any files that you find. With the `-exec` option, the command you use is executed on every file found, without stopping to ask if that's okay. The `-ok` option stops at each matched file and asks whether you want to run the command on it.

The advantage of using `-ok` is that, if you are doing something destructive, you can make sure that you okay each file individually before the command is run on it. The syntax for using `-exec` and `-ok` is the same:

```
$ find [options] -exec command {} \;
$ find [options] -ok command {} \;
```

With `-exec` or `-ok`, you run `find` with any options you like in order to find the files you are seeking. Then you enter the `-exec` or `-ok` option followed by the command you want to run on each file. The set of curly braces indicates where on the command line to read in each file that is found. Each file can be included in the command line multiple times if you like. To end the line, you need to add a backslash and semicolon (`\;`). Here are some examples:

- This command finds any file named `passwd` under the `/etc` directory and includes that name in the output of an `echo` command:

```
$ find /etc -iname passwd -exec echo "I found {}" \;
I found /etc/pam.d/passwd
I found /etc/passwd
```

- The following command finds every file under the `/usr/share` directory that is more than 5MB in size. Then it lists the size of each file with the `du` command. The output of `find` is then sorted by size, from largest to smallest. With `-exec` entered, all entries found are processed, without prompting:

```
$ find /usr/share -size +5M -exec du {} \; | sort -nr
116932  /usr/share/icons/HighContrast/icon-theme.cache
69048   /usr/share/icons/gnome/icon-theme.cache
20564   /usr/share/fonts/cjkuni-uming/uming.ttc
```

- The `-ok` option enables you to choose, one at a time, whether each file found is acted upon by the command you enter. For example, you want to find all files that belong to `joe` in the `/var/allusers` directory (and its subdirectories) and move them to the `/tmp/joe` directory:

```
# find /var/allusers/ -user joe -ok mv {} /tmp/joe/ \;
< mv ... /var/allusers/dict.dat > ? y
< mv ... /var/allusers/five > ? y
```

Notice in the preceding code that you are prompted for each file that is found before it is moved to the /tmp/joe directory. You would simply type **y** and press Enter at each line to move the file, or just press Enter to skip it.

For more information on the find command, enter **man find**.

Searching in files with grep

If you want to search for files that contain a certain search term, you can use the grep command. With grep, you can search a single file or search a whole directory structure of files recursively.

When you search, you can have every line containing the term printed on your screen (standard output) or just list the names of the files that contain the search term. By default, grep searches text in a case-sensitive way, although you can do case-insensitive searches as well.

Instead of just searching files, you can also use grep to search standard output. So, if a command turns out lots of text and you want to find only lines that contain certain text, you can use grep to filter just want you want.

Here are some examples of grep command lines used to find text strings in one or more files:

```
$ grep desktop /etc/services
desktop-dna    2763/tcp              # Desktop DNA
desktop-dna    2763/udp              # Desktop DNA

$ grep -i desktop /etc/services
sco-dtmgr      617/tcp               # SCO Desktop Administration Server
sco-dtmgr      617/udp               # SCO Desktop Administration Server
airsync        2175/tcp              # Microsoft Desktop AirSync Protocol
...
```

In the first example, a grep for the word desktop in the /etc/services file turned up two lines. Searching again, using the -i to be case-insensitive (as in the second example), there were 29 lines of text produced.

To search for lines that don't contain a selected text string, use the -v option. In the following example, all lines from the /etc/services file are displayed except those containing the text tcp (case-insensitive):

```
$ grep -vi tcp /etc/services
```

To do recursive searches, use the -r option and a directory as an argument. The following example includes the -l option, which just lists files that include the search text, without showing the actual lines of text. That search turns up files that contain the text peerdns (case-insensitive).

```
$ grep -rli peerdns /usr/share/doc/
/usr/share/doc/dnsmasq-2.66/setup.html
/usr/share/doc/initscripts-9.49.17/sysconfig.txt
...
```

The next example recursively searches the /etc/sysconfig directory for the term root. It lists every line in every file beneath the directory that contains that text. To make it easier to have the term root stand out on each line, the --color option is added. By default, the matched term appears in red.

```
$ grep -ri --color root /etc/sysconfig/
```

To search the output of a command for a term, you can pipe the output to the grep command. In this example, I know that IP addresses are listed on output lines from the ip command that include the string inet, so I use grep to display just those lines:

```
$ ip addr show | grep inet
inet 127.0.0.1/8 scope host lo
inet 192.168.1.231/24 brd 192.168.1.255 scope global wlan0
```

Summary

Being able to work with plain-text files is a critical skill for using Linux. Because so many configuration files and document files are in plain-text format, you need to become proficient with a text editor to use Linux effectively. Finding filenames and content in files are also critical skills. In this chapter, you learned to use the locate and find commands for finding files and grep for searching files.

The next chapter covers a variety of ways to work with processes. There, you learn how to see what processes are running, run processes in the foreground and background, and change processes (send signals).

Exercises

Use these exercises to test your knowledge of using the vi (or vim) text editor, commands for finding files (locate and find), and commands for searching files (grep). These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. Copy the /etc/services file to the /tmp directory. Open the /tmp/services file in vim, and search for the term WorldWideWeb. Change that to read World Wide Web.
2. Find the following paragraph in your /tmp/services file (if it is not there, choose a different paragraph) and move it to the end of that file.

```
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1700, "Assigned Numbers" (October 1994). Not all ports
# are included, only the more common ones.
```

3. Using `ex` mode, search for every occurrence of the term `tcp` (case-sensitive) in your `/tmp/services` file and change it to `WHATEVER`.
4. As a regular user, search the `/etc` directory for every file named `passwd`. Redirect error messages from your search to `/dev/null`.
5. Create a directory in your home directory called `TEST`. Create files in that directory named `one`, `two`, and `three` that have full read/write/execute permissions on for everyone (user, group, and other). Construct a `find` command to find those files and any other files that have write permission open to "others" from your home directory and below.
6. Find files under the `/usr/share/doc` directory that have not been modified in more than 300 days.
7. Create a `/tmp/FILES` directory. Find all files under the `/usr/share` directory that are more than 5MB and less than 10MB and copy them to the `/tmp/FILES` directory.
8. Find every file in the `/tmp/FILES` directory, and make a backup copy of each file in the same directory. Use each file's existing name, and just append `.mybackup` to create each backup file.
9. Install the `kernel-doc` package in Fedora or Red Hat Enterprise Linux. Using `grep`, search inside the files contained in the `/usr/share/doc/kernel-doc*` directory for the term `e1000` (case-insensitive) and list the names of the files that contain that term.
10. Search for the `e1000` term again in the same location, but this time list every line that contains the term and highlight the term in color.